Introduction to the Virtual Drum Kit

Imagine This:

Having your own invisible and portable drum kit that not only takes up far less space, but also costs ten times less than a real one. With virtual reality no longer a distant dream and technology headed towards its implementation in all applications, a virtual interface for musical instruments is inevitable. This is the focus of “Tap That” – creating a cost-effective, compact and enjoyable drumming experience by means of a virtual drum kit.

Past Work

The first virtual drum kit was created by a French group (www.virtual-drums.com). Their implementation uses two webcams for 3D spatial analysis and extensive C++ coding. There are several other virtual drum kits available on the internet, but they must be controlled with either a mouse or a keyboard, which detracts from the drumming experience. The drum kits developed by the French group and our group give the user a real feeling of playing the drums while actually being virtual.

Our Goal

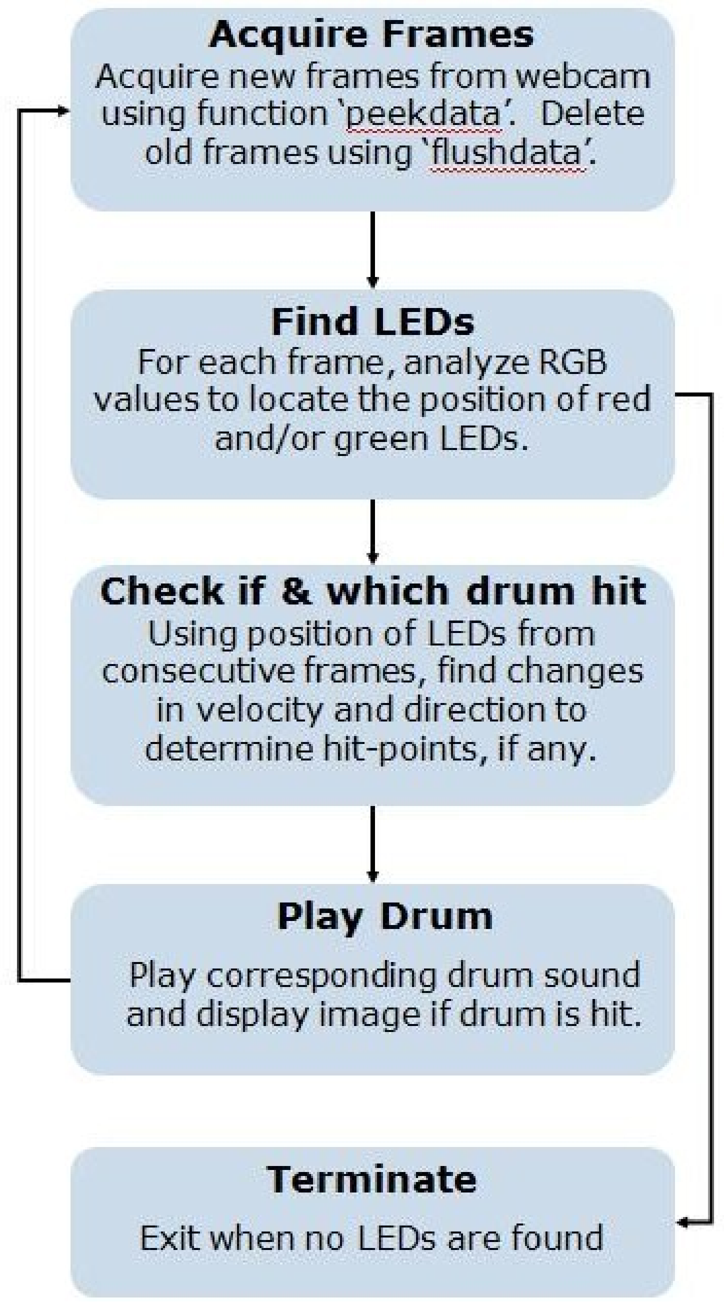
This project aims at creating an affordable virtual drum kit for amateur drummers using readily available materials. Music enthusiasts can “Tap That” with this cost-effective replacement of a real drum kit. This instrument requires a web camera, Matlab 6.0 or higher, two LED lights (mounted on drumsticks for a more authentic experience) and a computer.

Summary

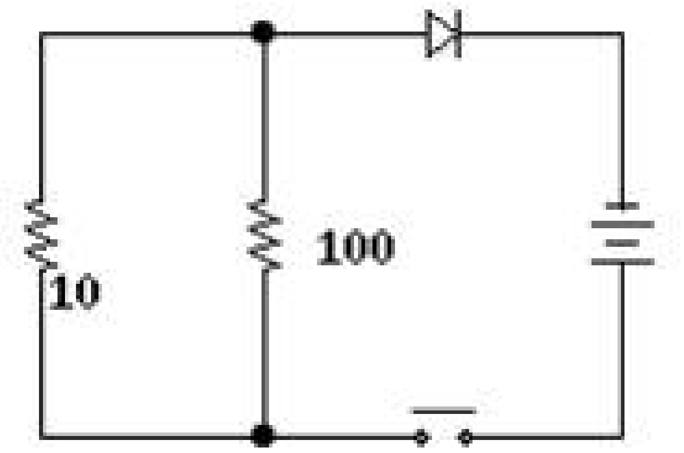
This project is based on a 2-dimensional tracking system. It tracks the LEDs in two dimensions and uses velocity computations to determine when and where the drum was hit. The user positions himself or herself on a chair in front of the webcam and runs the program. An animated image on the screen gives the user the approximate positions of the drums. As the user moves the LEDs in the air, pretending to hit real drums, Matlab calculates the position of the LEDs by carrying out a frame-by-frame analysis of the video in real-time. The velocities of the LEDs are calculated, and when the LED starts moving upwards instead of downwards, it means that a drum has been hit. Based on the position of the strike, the corresponding drum’s sound will be produced while the animation gives the user feedback about which drum they hit.

A General Approach to Building Your Own Virtual Drum Kit

Flow Chart


Acquire Frames
Acquire new frames from webcam using function 'peekdata'. Delete old frames using 'flushdata'.
Find LEDs
For each frame, analyze RGB values to locate the position of red and/or green LEDs.
Check if & which drum hit
Using position of LEDs from consecutive frames, find changes in velocity and direction to determine hit-points, if any.
Play Drum
Play corresponding drum sound and display image if drum is hit.
Terminate
Exit when no LEDs are found

The LED Drum Sticks



The LED drumsticks consist of a simple LED circuit mounted on narrow pipes. Instead of using a battery holder and a switch, we attached a pen flashlight to the circuit. The program also requires that one of the LEDs be red, and the other green. The circuit consisted of a super-bright LED, 9.11 ohm of resistance and a 3 V source. A lower resistance will make the LED brighter, making its detection better. Alternatively, the user can obtain LED sticks on the internet for $5 each. When the switch at the end of the flashlight is turned on, the LEDs will light up and serve as detection points of the drumsticks. The program stops running if you switch off the LED drum sticks.

The only limitation posed by the LED drumsticks is that they have to be used in a dimly lit area to be detected without errors.

Now that you have your drumsticks, you are all set on the hardware and need to implement the software!

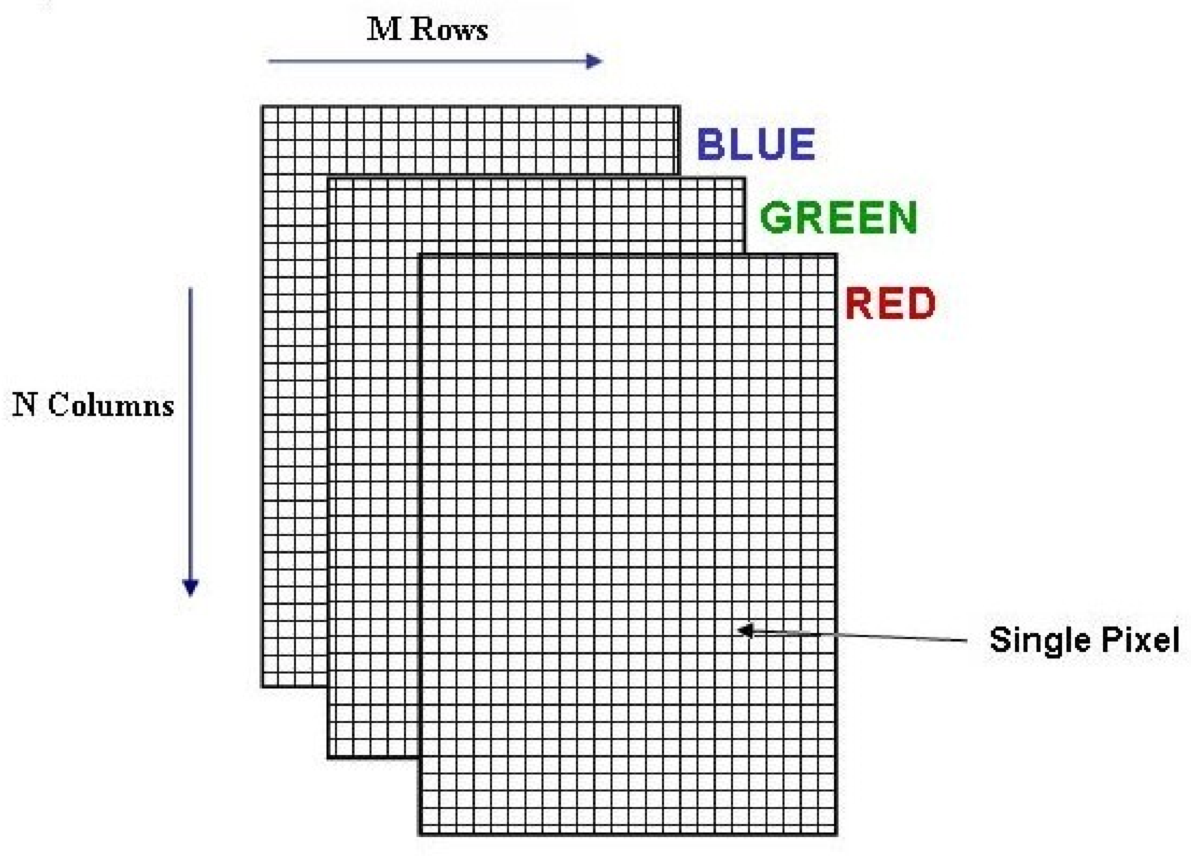
Webcam Color Tracking in Matlab

Back ground: Image Processing in Matlab

What composes an image?

Each image is composed of an array of M*N pixels (contraction of “picture element”) with M rows and N columns of pixels. Each pixel contains a certain value for red, green and blue. Varying these values for red, green, blue (RGB) we can get almost any color.

Image Storage in Matlab



Color Detection

The RGB format is a practical method to represent color images. Matlab creates three matrices (or three M x N arrays) with each matrix representing normalized components of red, green or blue to read and store each of the frames of the video. Any pixel's color is determined by the combination of Red, Green and Blue values stored in the three matrices at that pixel's location. This is how Matlab reads and manipulates .jpg files.

Images:

```
picture (row, column, rgb value)
```

For example, picture (12, 78, 1) corresponds to the red component of the pixel at row 12 and column 78; picture(12, 78, 2) corresponds to the green component of the pixel and picture(12, 78, 3) gives us the blue component of the pixel at that location.

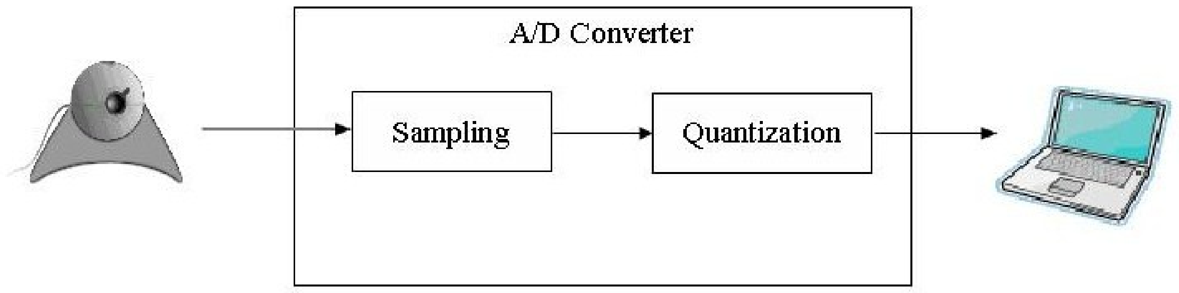
Videos:

```
frames(row, column, rgb value, frame)
```

Videos have an extra dimension for the frame number. So frames(12,78,1,5) would correspond to the red component of the pixel in the 12th row and 78th column of the 5th frame. To get the entire frame, we could just say frames(:,:,:,5).

Acquiring Images From The Webcam in Matlab

Digitization of the Images



How you acquire the images from the camera depends a lot on what software you are using to implement it. Creating the interface between the computer and the drum using a lower level language like C or C++ will give you a lot of flexibility, but it will also involve a lot of work and background knowledge. Fortunately for us, Matlab’s Image Acquisition Toolbox has a variety of simple functions that can be used to create the interface. The next few paragraphs will describe these functions and how we used them in some detail.

For Matlab to recognize the video camera you have to create it as an object using the command `obj=videoinput(‘winvideo’)`. Matlab will automatically find the webcam connected to your computer. Once it is an object in your workspace you can edit its settings, such as the number of frames per second, to optimize it for your project. `preview()` and `getframe()` are two useful functions for determining if the camera has been positioned properly. The first allows you to see what the camera sees, without collecting any data from it, and the second acquires a single snapshot and stores it as in image.

Now we can start recording the video. With the `start(obj)` command, the camera will be triggered and will start to collect as many frames as specified by the **FramesPerTrigger** property. These frames are stored in the camera’s buffer memory. These frames are stored in the 4D array as described above.

To retrieve them from the buffer you could use either the `getdata()` function or the `peekdata()` function.

getdata(obj);

When this statement is encountered, the program retrieves all the frames acquired at the last trigger and then empties the buffer.

peekdata(obj,M);

This function allows us to “peek” at the last M frames collected by the camera. The frames are copied, but not cleared, from the camera’s buffer into Matlab’s workspace.

Both functions take about the same amount of time to run. Interestingly, the number of frames acquired at a time does not affect the execution time much. It takes about as long to acquire 1 frame as it does to acquire 50. Therefore, to make the program more efficient, we should collect large chunks of data at a time.

To make sure that we don’t miss any of the action while the user is waving the drumsticks around in front of the camera, we should also make sure that we can get all the frames from when the program starts running till the user turns of the LEDs. Due to memory limitations, our computer could collect maximum of 240 frames, or about 8 seconds of the video, which is clearly not enough. It is unlikely that your computer could do much better.

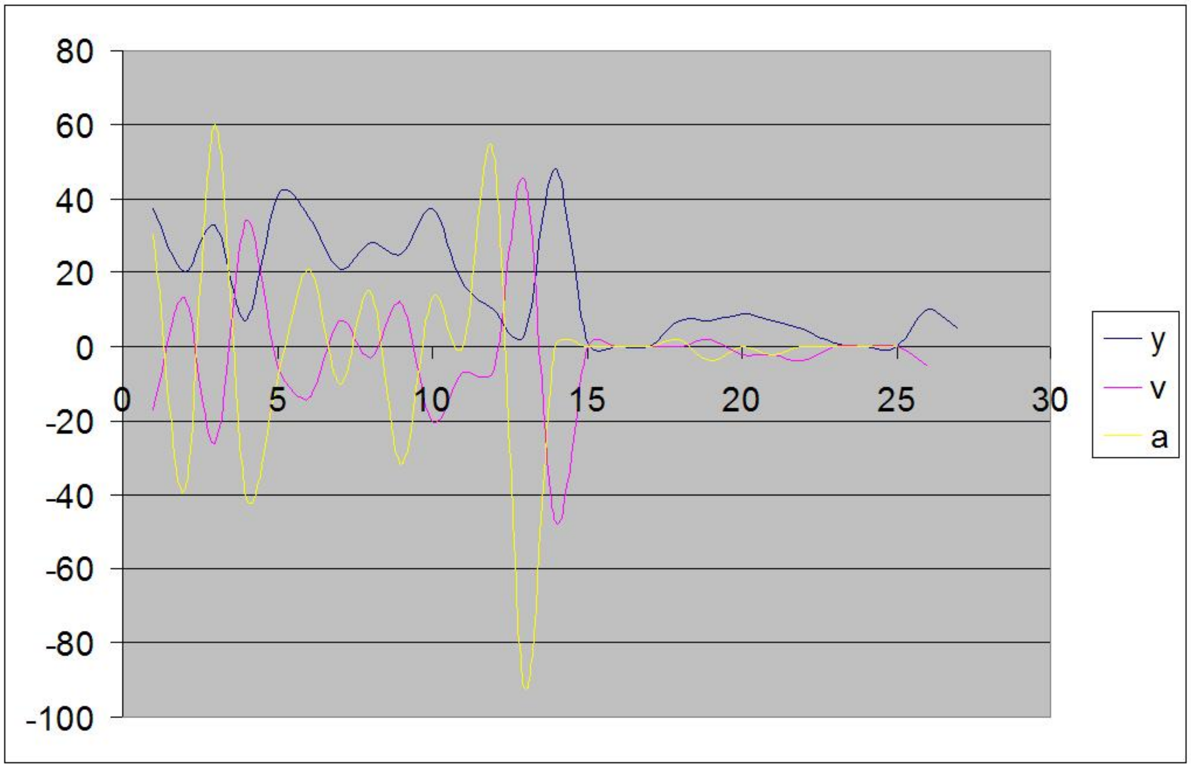
Implementation: Detecting a Hit

Hitting the Drums: A Velocity Computational Approach

You now have the positions of the LEDs in every frame. Our goal here is to detect when and which drums were hit so that the corresponding drums can produce their sounds.

Determining when a drum was hit

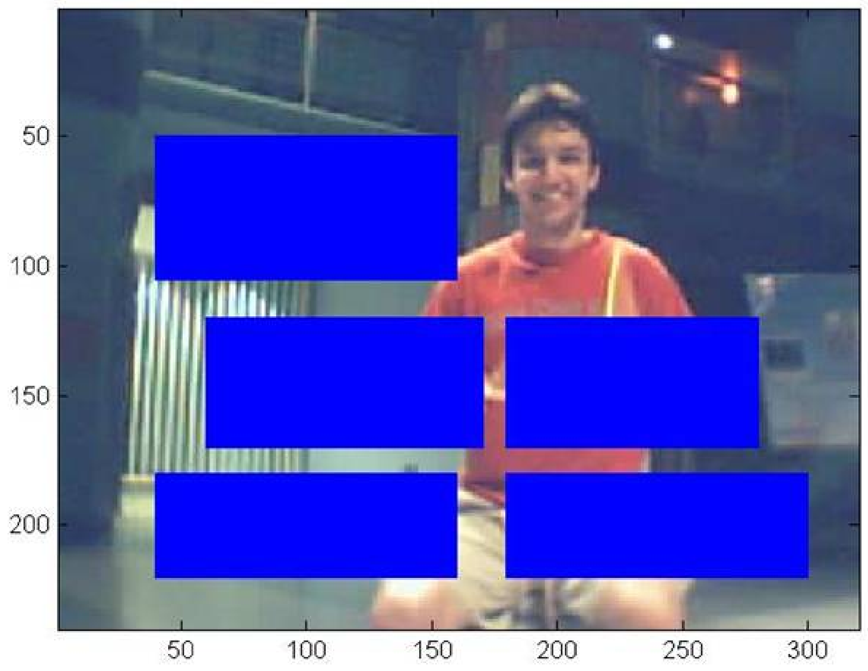
There are various innovative ways of tracking when the drums are hit. We chose to use a velocity computational approach due to its speed. Using this approach, you can look at the displacement of the LEDs in consecutive frames and calculate the velocity. Using basic physics, if the velocity changes from positive to negative (note that our coordinate system has the positive side of the y-axis pointing downwards) it implies a change in direction of the drumstick, or that a drum was hit. You should be able to figure out approximately which frame the drum was hit in. Since the hit point is returned in terms of two-dimensional coordinates, you can figure out which drum(s) are hit.
Variation in velocity with the movement of the LEDs



See if you can guess when drums were hit.

Determining which drum was hit

Position of the Drums



[1: Cymbal; 2: BigTom; 3: Small Tom; 4: Floor Tom; 4: Snare]

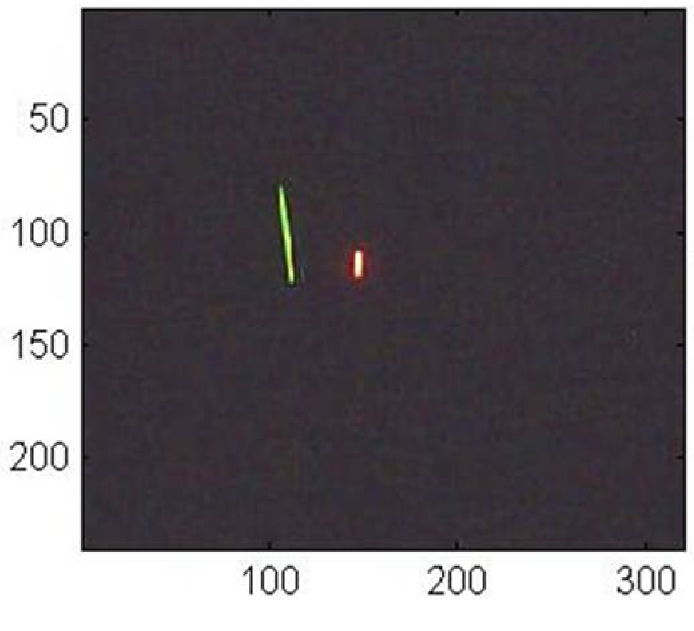
The frame should be broken down into rectangles with assigned x and y coordinates where the drums are positioned. Our drum kit consists of the Crash Cymbal, the Big Tom, the Small Tom, the Floor Tom and the Snare. We check to see if the hit-point falls in the space assigned for any of the drums, and if it does, we move on to producing the sound and displaying the image on the computer screen. However, if the hit point is outside the frame, it is taken as a missed hit.

Implementation: Detecting the LEDs

RGB in Each Individual Frame

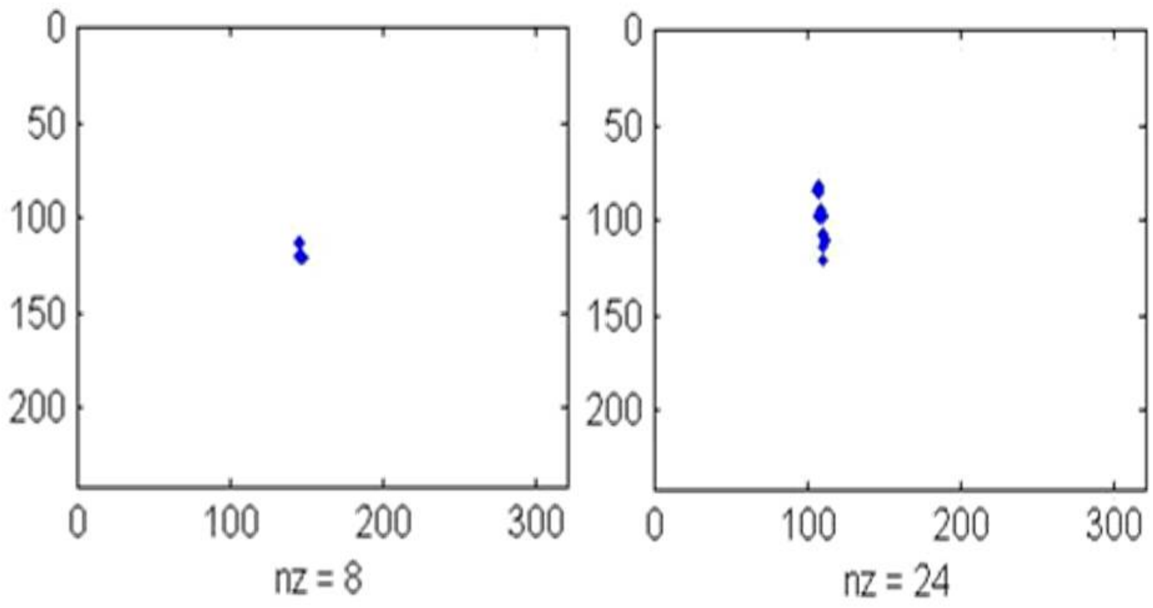
Given the acquired frames, the first step is to locate the LEDs. The color of the LEDs can be described by the intensity of red, green and blue. These are represented by values between 0 and 255 (going from null intensity to full intensity) in the RGB matrices for each image. We can detect the LEDs by setting threshold values for Red, Green and Blue. For example, to find the Red LED, you are looking for the part of the frame with high Red values and low Blue and Green values. We would recommend using the data cursor to analyze RGB values of the LED in the colormap. Play around with threshold values until you get the right thresholds, i.e., only the LED you are trying to detect shows up.

A Sample Frame



Spy Plots of the LEDs

Red LED Green LED



In our project, we used the find function to look at the red, green and blue matrices separately and set ones wherever the conditions were satisfied. We multiply the three matrices of ones and zeros to produce one final matrix with ones where all three color thresholds are satisfied and the LED is detected.

Turn Off the LEDs

If the LEDs are not detected for 75% of the frames, it means that the LEDs are either turned off or not in the frame at all, and the program stops running. However, if the LEDs have been found, you can move on to detecting when the drums are hit.

Producing Drum Sounds and Displaying the Drum

Once we have determined which drum has been hit, we play the appropriate sound and display the corresponding image. The drum sounds and images are preloaded in separate .mat files at the beginning of the program using the load command to expedite execution.

```
load drumsounds
```

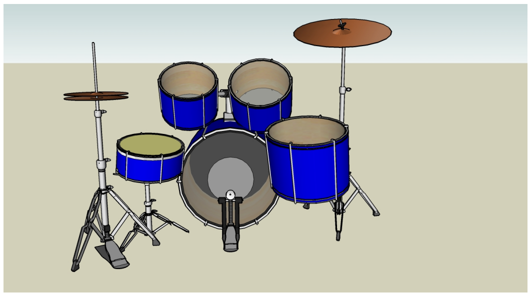
```
load drumpics
```

Drum Sounds:

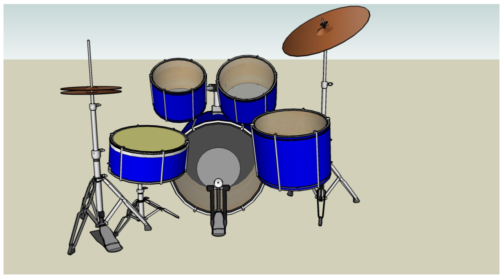
`drumpics.mat` was created by importing drum samples in the wav format using `wavread()` and `wavadd()`. `wavread('wavfile')` is a Matlab function that stores wav files as an Nx2 matrix. `wavadd(sound1,sound2)` is a function that we created to allow the simultaneous play of two different sounds. For example, if both the snare and the crash cymbal were hit at the same time, we zero pad the shorter matrix and combine them as one sound matrix. In the main program, the Matlab function `soundsc(sound, fs)` plays the corresponding sound file, where 'fs' is the sampling rate. Our wav files were sampled at 22 kHz to allow for smaller sound variables without affecting the sound quality too much. All of our drum samples were obtained free online from a Yamaha 9000 drum kit.

Drum Pictures:

`drumsounds.mat` was created by importing jpeg files using the `imread()` Matlab command to store the images as variables. The images were created from Google SketchUp, a free 3D modeling program available online. Different views were created to simulate the feel of animation.



Main view



Crash cymbal and snare were hit

Implementation: Code

To get a better understanding for how we implemented the drum kit, you might want to have a look at our Matlab code.

The code is available at:
http://www.geocities.com/tapthat_virtualdrum/tapthat.html

Essential Files:

proj_301.m- This is a wrapper function to set up video acquisition and collect frames from the camera.

detectcolor.m- This function checks for presence of the LED.

coord.m- This function finds the position of the LED in each frame, computes its velocity and hit-point.

drumdet.m- This function determines which drum was hit.

drumdisp.m- It produces the sound and displays the image corresponding to the drum that was hit.

Files for debugging and setting up the system:

adjust.m- This is mainly for the user’s convenience. It helps adjust the position of the user with respect to the drums.

checkframe.m- This can be used to verify that the color detection works properly for a particular frame.

Results of the Virtual Drum Kit

Results

While there are many ways to approach the virtual drum kit, we are pleased with the outcome of our implementation using one webcam and velocity computation. We found that users enjoyed the drum kit more after familiarizing themselves with the set-up.

Advantages

- High precision in detecting where the drums are hit. Our results showed that less than two hits were missed in a minute’s worth of playing.
- Affordable solution for drumming enthusiasts.
- Easily Transportable.
- Runs in real time without a significant delay with Matlab set to high priority.
- Easy to modify the Matlab code to make the system customizable.
- Animated interface gives a visual representation of drums as feedback hit instead of just bare audio.
- Absence of physical drums makes the drummer visible to the audience.

Limitations

- Slight delay due to processing power of the computer.
- Requires a dimly lit area for easy LED detection.

Future Work for the Virtual Drum Kit

Conclusion

While the virtual drum kit is not advanced enough to replace a real drum kit for a professional drummer, it is an enjoyable alternative for anyone aspiring to learn how to play the drums or amateur music enthusiasts looking for an affordable solution.

Further improvements in our implementation could lead to a drum kit that is almost as good as a real drum kit.

Room for Improvement

- Addition of the bass drum, hi hat, ride cymbals for a fuller drum kit.
- Incorporation of a change in volume based on the acceleration with which drums are hit.
- Higher frame rate and more powerful machines to overcome the delay and visual latency.
- Adding customizability to allow the user to modify the positions of the drums, and the sounds that are played.

Future Work

The concept of LED tracking forms the base of many useful implementations. More specifically, our Matlab code can be easily modified to implement a virtual whiteboard or any other percussion instrument.

With a sense of adventure and some degree of technological innovation and creativity, virtual musical instruments can replace expensive physical musical instruments while delivering comparable results.

Team Members and Acknowledgments

"Tap That" was brought to you by Janice Chow, Tanwee Misra and Kriti Charan.

Kriti Charan maintained the caffeinated group's sanity and ensured that we stuck to the cynical 6-point project plan as outlined in Mr. Dye's office. She connected the webcam to the computer. Not just by plugging the USB cord in, but by writing proj_301.m. She also channeled her fear of insects into debugging the code. Basically she was the project guru.

Janice Chow supplied us with free coffee and was responsible for deciding where the user had to hit to make the program react in a certain way, and what those reactions should be. In other words she wrote drumdisp.m and drumdet.m, and by nefarious means, obtained all the images and sounds that were required. She was also a great test subject to see if our program was working or not and tremendously enjoyed playing with Google's ketchup.

Tanwee Misra kept us entertained with her French and was responsible for the program's ability to see color and use that to figure out what the test subject was doing with the drumsticks. In other words she wrote coord.m and detectcolor.m. She also encouraged group projects by working with Janice to make the poster, and Kriti to make the drumsticks.

We are grateful to **Mr. Michael Dye** and **Dr. James Young** for all the help they provided as we made and repaired the drumsticks. We are also grateful to **Dr. Kronister** for unknowingly providing us with so much.

We would also like to thank **Dr. B (aka RichB)** and **Mr. D (aka Mark Davenport)** for all the help, support and guidance that they have provided at every step along the way.

Special thanks to all our **rock star fans** who tried our drum kit on the day of the presentation.